Implementasi Tanda Tangan Digital untuk Verifikasi *Mod* Cities: Skylines II

Bastian H. Suryapratama - 13522034[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522034@std.stei.itb.ac.id, [2]bastian.hensur@gmail.com

***Abstrak*—Industri permainan video modern sangat bergantung pada konten buatan komunitas (*modding*) untuk memperkaya pengalaman bermain. Cities: Skylines II, sebagai gim simulasi pembangunan kota yang populer, mendukung modifikasi logika yang mendalam melalui berkas *Dynamic Link Library* (.dll). Namun, arsitektur ini membuka celah keamanan yang serius karena memungkinkan eksekusi kode sembarangan. Risiko ini terbukti nyata dengan adanya insiden penyusupan perangkat lunak perusak (*malware*) pada *mod* "Traffic" yang menargetkan pencurian data sensitif pengguna. Makalah ini bertujuan merancang dan mengimplementasikan mekanisme pengamanan distribusi *mod* Cities: Skylines II menggunakan Tanda Tangan Digital. Sistem yang dibangun memanfaatkan algoritma kriptografi kunci publik RSA dengan panjang kunci 2048-bit dan skema *padding* PSS (*Probabilistic Signature Scheme*) untuk menjamin keamanan terhadap pemalsuan. Selain itu, fungsi hash SHA-256 diterapkan untuk memverifikasi integritas keseluruhan konten dalam arsip ZIP *mod*. Implementasi dilakukan menggunakan bahasa pemrograman Python, menghasilkan alat verifikasi yang mampu mendeteksi modifikasi ilegal dan menjamin autentikasi pengirim sebelum *mod* dijalankan oleh pengguna.**

***Kata Kunci*—Cities: Skylines II, Keamanan *Mod*, RSA-PSS, SHA-256, Tanda Tangan Digital.**

I. Pendahuluan

Industri permainan video modern tidak lagi hanya bergantung pada konten yang disediakan secara statis oleh pengembang resmi, melainkan juga sangat didorong oleh kreativitas komunitas melalui modifikasi permainan atau yang lebih dikenal dengan istilah *modding*. Salah satu contoh paling terkenal dalam genre simulasi pembangunan kota adalah Cities: Skylines II. Dirilis oleh Colossal Order dan diterbitkan oleh Paradox Interactive pada tahun 2023, gim ini dirancang dengan arsitektur yang terbuka, melanjutkan kesuksesan pendahulunya yang memiliki ribuan modifikasi buatan komunitas.

Dalam ekosistem Cities: Skylines II, *mod* memiliki peran krusial dalam memperkaya pengalaman bermain, mulai dari penambahan aset bangunan, perbaikan logika lalu lintas (*traffic logic*), hingga modifikasi antarmuka pengguna. Namun, arsitektur teknis *mod* pada gim ini membawa risiko keamanan tersendiri. Cities: Skylines II dibangun menggunakan *game engine* Unity dan bahasa pemrograman C#. Hal ini memungkinkan para *modder* (pembuat *mod*) untuk mendistribusikan *mod* mereka dalam bentuk berkas kode biner atau *Dynamic Link Library* (.dll) yang dikemas dalam sebuah arsip (biasanya format .zip). Berbeda dengan modifikasi yang hanya mengganti tekstur gambar, berkas .dll ini dieksekusi langsung oleh prosesor saat gim berjalan. Hal ini memberikan akses yang sangat luas bagi kode *mod* terhadap sistem operasi pengguna.

Meskipun fleksibilitas ini mendorong inovasi fitur yang masif, hal ini juga membuka celah keamanan siber yang serius berupa eksekusi kode sembarangan (*arbitrary code execution*). Kebebasan dalam mendistribusikan berkas berekstensi .dll tanpa mekanisme verifikasi kriptografis yang ketat pada sisi klien menciptakan risiko masuknya kode berbahaya (*malicious code*). Masalah ini diperparah dengan terfragmentasinya platform distribusi *mod*. Pengguna dapat mengunduh mod dari platform resmi seperti Paradox Mods, maupun dari repositori pihak ketiga yang dikelola komunitas.

Risiko keamanan ini telah terbukti menjadi ancaman nyata. Pada akhir tahun 2024, komunitas Cities: Skylines II menghadapi insiden keamanan siber yang signifikan. Salah satu *mod* populer bernama "Traffic", yang bertujuan memperbaiki sistem lalu lintas dalam gim, dilaporkan telah disusupi oleh perangkat lunak perusak (*malware*). Berdasarkan pernyataan resmi dari Paradox Interactive pada 31 Oktober 2024, seorang pelaku ancaman (*threat actor*) berhasil memperbarui *mod* tersebut dengan berkas .dll yang berbahaya. *Mod* yang telah disusupi *malware* tersebut kemudian didistribusikan kembali kepada pengguna seolah-olah merupakan pembaruan resmi dari pengembang *mod*. Tanpa disadari oleh pengguna, *mod* tersebut mengandung *script* yang menargetkan pencurian data sensitif, khususnya kredensial dompet kripto (*crypto wallet*) seperti Exodus.

Insiden "Traffic" *mod* ini menyoroti kelemahan

fundamental dalam rantai distribusi *mod* saat ini, yaitu ketiadaan jaminan integritas dan autentikasi yang dapat diverifikasi secara mandiri oleh pengguna akhir. Saat seorang pemain mengunduh arsip *mod* berbentuk ZIP, mereka tidak memiliki cara yang pasti untuk membedakan antara berkas asli buatan pengembang terpercaya dengan berkas yang telah dimanipulasi di tengah jalan (*Man-in-the-Middle*) atau di server penyimpanan. Mekanisme keamanan konvensional seperti *scanning malware* seringkali terlambat mendeteksi varian *malware* baru.

Oleh karena itu, diperlukan sebuah pendekatan kriptografis untuk mengamankan distribusi *mod*. Makalah ini mengusulkan implementasi skema Tanda Tangan Digital (*Digital Signature*) sebagai solusi preventif. Tanda tangan digital memanfaatkan pasangan kunci asimetris (kunci publik dan kunci privat) serta fungsi *hash* satu arah. Dalam skema yang diusulkan, setiap pembuat *mod* akan menandatangani keseluruhan isi arsip ZIP *mod* mereka menggunakan kunci privat. Pengguna, melalui sistem verifikasi otomatis, dapat menggunakan kunci publik pembuat mod untuk memastikan dua hal penting: (1) Integritas Data, menjamin bahwa tidak ada satu bit pun dalam arsip ZIP yang berubah (misalnya penyisipan virus ke dalam .dll) sejak ditandatangani, dan (2) Autentikasi Pengirim, menjamin bahwa berkas tersebut benar-benar berasal dari pembuat *mod* yang sah.

Makalah ini akan memaparkan simulasi teknis proses pengamanan distribusi *mod* Cities: Skylines II yang diimplementasikan menggunakan bahasa pemrograman Python. Sistem ini dibangun menggunakan algoritma kriptografi kunci publik RSA dengan panjang kunci 2048-bit yang dipadukan dengan skema padding PSS (*Probabilistic Signature Scheme*) untuk memberikan lapisan keamanan yang kuat terhadap pemalsuan tanda tangan. Selain itu, fungsi *hash* SHA-256 diterapkan untuk membuat ringkasan pesan (*message digest*) dari keseluruhan konten arsip ZIP *mod*, sehingga integritas seluruh berkas di dalamnya dapat diverifikasi secara menyeluruh.

II. Dasar Teori

A. Kriptografi dan Keamanan Informasi

Secara etimologis, kata kriptografi berasal dari bahasa Yunani, yaitu *kryptos* yang berarti "tersembunyi" dan *graphein* yang berarti "tulisan". Dalam definisi klasik yang dikemukakan oleh Schneier (1996), kriptografi digambarkan sebagai ilmu dan seni untuk menjaga keamanan pesan (*secure message*). Namun, seiring dengan pesatnya perkembangan teknologi informasi, definisi ini telah bertransformasi. Di era digital modern, kriptografi tidak lagi sekadar seni menyandikan huruf, melainkan telah menjadi ilmu pasti yang berbasis pada teknik matematika yang kompleks (seperti teori bilangan dan aljabar) untuk menangani keamanan informasi digital. Dalam konteks distribusi perangkat lunak atau modifikasi gim (*game mods*) seperti pada Cities: Skylines II, kriptografi menjadi fondasi utama untuk menciptakan ekosistem pertukaran data yang aman di jaringan publik (internet) yang tidak aman.

Sistem kriptografi yang kuat harus mampu memenuhi beberapa aspek fundamental keamanan informasi untuk menanggulangi ancaman siber. Dalam skenario distribusi *mod*, beberapa aspek tersebut digambarkan sebagai berikut:

1. Kerahasiaan (*Confidentiality*)
   Aspek ini menjamin bahwa informasi hanya dapat diakses atau dibaca oleh pihak yang memiliki otorisasi. Meskipun *mod* gim umumnya bersifat *open-source* dan didistribusikan secara gratis kepada publik (sehingga kerahasiaan isi berkas *mod* bukan prioritas utama), aspek ini tetap memegang peranan vital dalam perlindungan kunci. Kunci privat (*private key*) milik pengembang *mod* harus dijaga kerahasiaannya secara mutlak agar tidak dicuri oleh pihak tidak bertanggung jawab yang ingin melakukan pemalsuan identitas.
2. Integritas Data (*Data Integrity*)
   Ini adalah aspek yang paling krusial dalam distribusi *mod* Cities: Skylines II. Integritas menjamin bahwa data yang diterima oleh pengguna (*gamer*) adalah persis sama dengan data yang dikirimkan oleh pengembang *mod* (*modder*). Tidak boleh ada perubahan sedikitpun, baik itu penambahan, pengurangan, maupun manipulasi bit selama proses transmisi data dari server ke komputer pengguna. Kegagalan pada aspek ini dapat berarti berkas *mod* telah disusupi kode berbahaya (*malware*) atau rusak (*corrupted*) saat pengunduhan.
3. Autentikasi (*Authentication*)
   Aspek ini berkaitan dengan verifikasi identitas pengirim. Di tengah banyaknya repositori *mod* pihak ketiga, pengguna seringkali kesulitan membedakan pengembang asli dan peniru. Sistem kriptografi harus mampu membuktikan bahwa sebuah *mod* yang diklaim berasal dari "Modder A" benar-benar dibuat oleh "Modder A", bukan oleh penyerang yang melakukan teknik penyamaran (*spoofing*) untuk menyebarkan program berbahaya atas nama orang lain.
4. Nirsangkal (*Non-repudiation*)
   Aspek ini mencegah entitas pengirim untuk menyangkal bahwa mereka telah mengirimkan pesan atau data tersebut di kemudian hari. Jika sebuah *mod* yang ditandatangani secara digital terbukti berisi *malware* atau merusak komputer pengguna, pembuat *mod* (sebagai pemilik kunci privat yang sah) tidak dapat menyangkal bahwa tanda tangan tersebut berasal dari kuncinya. Hal ini memberikan jaminan akuntabilitas dalam komunitas *modding*.

Untuk memahami mekanisme kerja tanda tangan digital, terdapat beberapa terminologi teknis yang perlu dipahami dalam konteks distribusi *mod*:

1. Pesan (*Message*) / *Plaintext*

Dalam kriptografi klasik, bagian ini merujuk pada teks asli yang dapat dibaca. Dalam konteks makalah ini, *plaintext* adalah keseluruhan berkas arsip *mod* (format .zip) yang berisi kode .dll dan aset gim yang belum diproses secara kriptografi.

2. *Ciphertext*
   Merupakan pesan yang telah disandikan. Dalam skema tanda tangan digital, *ciphertext* merujuk pada "Tanda Tangan" itu sendiri, yang merupakan hasil enkripsi dari nilai *hash* berkas *mod*.
3. Enkripsi dan Dekripsi
   Enkripsi adalah proses matematis mengubah plainteks menjadi *ciphertext* menggunakan kunci tertentu, sedangkan dekripsi adalah proses kebalikannya.
4. Kunci (*Key*)
   Parameter rahasia atau publik yang digunakan dalam algoritma kriptografi. Keamanan sistem modern tidak bergantung pada kerahasiaan algoritma, melainkan pada kerahasiaan kunci ini.

Berdasarkan mekanismenya, algoritma kriptografi dikategorikan menjadi tiga jenis utama:

1. Kriptografi Simetris (*Symmetric-key Cryptography*)
   Algoritma ini menggunakan satu kunci yang sama untuk proses enkripsi (penguncian) dan dekripsi (pembukaan).
2. Kriptografi Asimetris (*Asymmetric-key Cryptography*)
   Dikenal juga sebagai Kriptografi Kunci Publik. Algoritma ini menggunakan sepasang kunci, yaitu Kunci Privat (untuk menandatangani) dan Kunci Publik (untuk memverifikasi). Ini adalah solusi yang cocok untuk distribusi *mod*. Pengembang menyimpan Kunci Privat untuk membuat tanda tangan digital, dan menyebarkan Kunci Publik secara bebas kepada komunitas. Pengguna dapat memverifikasi keaslian *mod* menggunakan Kunci Publik tanpa perlu mengetahui Kunci Privat pengembang. Algoritma RSA adalah contoh standar dari jenis algoritma ini.
3. Fungsi Hash (*Hash Function*)
   Berbeda dengan dua jenis algoritma sebelumnya, fungsi *hash* adalah algoritma fungsi satu arah (*one-way function*) yang mengubah input data sepanjang apapun menjadi string acak dengan panjang tetap (*fixed size*) yang disebut *digest* atau *fingerprint*. Algoritma ini, seperti SHA-256, sangat penting untuk menjamin integritas. Perubahan satu bit saja pada berkas *mod* akan mengubah nilai *hash* secara drastis (*avalanche effect*), sehingga sistem dapat mendeteksi modifikasi ilegal dengan cepat.

B. Kriptografi Kunci Publik (Asimetris)

Kriptografi kunci publik, atau sering disebut sebagai kriptografi asimetris, merupakan terobosan fundamental dalam sejarah keamanan informasi yang diperkenalkan oleh Whitfield Diffie dan Martin Hellman pada tahun 1976. Sebelum ditemukannya teknik ini, keamanan komunikasi sangat bergantung pada kriptografi simetris, di mana pengirim dan penerima harus berbagi kunci rahasia yang sama. Kelemahan utama dari metode simetris adalah masalah distribusi kunci (*key distribution problem*): bagaimana cara mengirimkan kunci rahasia tersebut kepada penerima tanpa diketahui oleh pihak ketiga? Jika kunci tersebut dicegat saat pengiriman, maka seluruh keamanan sistem akan runtuh.

Kriptografi asimetris memecahkan masalah tersebut dengan menggunakan sepasang kunci yang berbeda namun saling berpasangan secara matematis. Sepasang kunci ini memiliki hubungan unik di mana data yang dienkripsi oleh salah satu kunci hanya dapat didekripsi oleh kunci pasangannya. Mekanisme ini memungkinkan komunikasi yang aman dilakukan di saluran yang tidak aman tanpa perlu mempertukarkan kunci rahasia terlebih dahulu.

Dalam skema asimetris, setiap entitas (dalam kasus ini, pembuat *mod*) memiliki dua jenis kunci dengan fungsi yang spesifik:

1. Kunci Publik (*Public Key*)
   Sesuai namanya, kunci ini tidak bersifat rahasia. Kunci publik disebarkan secara luas kepada komunitas, diunggah ke *repository mod*, atau disertakan dalam paket instalasi. Dalam konteks verifikasi *mod* Cities: Skylines II, kunci publik digunakan oleh komputer pengguna (*gamer*) untuk melakukan verifikasi. Kunci ini berfungsi untuk memeriksa validitas tanda tangan digital yang melekat pada berkas *mod*. Siapapun dapat memiliki kunci publik ini, namun mereka tidak dapat gunakannya untuk memalsukan tanda tangan.
2. Kunci Privat (*Private Key*)
   Kunci ini bersifat sangat rahasia dan harus dijaga ketat oleh pemiliknya (pembuat *mod*). Kunci ini tidak boleh didistribusikan atau dikirimkan melalui jaringan. Fungsi utama kunci privat dalam sistem ini adalah untuk membangkitkan tanda tangan digital (*signing*). Hanya pemegang kunci privat yang sah yang dapat membuat tanda tangan yang valid untuk sebuah berkas *mod*. Jika kunci ini bocor, maka penyerang dapat membuat *mod* berbahaya dan menandatanganinya seolah-olah *mod* tersebut berasal dari pengembang asli.

Keamanan algoritma asimetris tidak didasarkan pada kerahasiaan algoritma itu sendiri, melainkan pada kesulitan komputasi untuk memecahkan masalah matematika tertentu. Konsep ini dikenal sebagai *Trapdoor One-Way Function*, yaitu sebuah fungsi matematika yang mudah dihitung dalam satu arah (misalnya perkalian), tetapi sangat sulit untuk dibalikkan (misalnya pemfaktoran) tanpa memiliki informasi khusus yang disebut "*trapdoor*" (kunci privat).

C. Tanda Tangan Digital (Digital Signature)

Tanda tangan digital adalah mekanisme kriptografi yang berfungsi sebagai tanda tangan ekuivalen digital dari tanda tangan basah pada dokumen fisik atau cap stempel resmi. Namun, tanda tangan digital menawarkan jaminan

keamanan yang jauh lebih tinggi. Jika tanda tangan basah pada kertas dapat dipalsukan dengan meniru goresan tinta atau dipindahkan (disalin) ke halaman lain, tanda tangan digital memiliki sifat unik, yaitu melekat secara matematis pada setiap bit data yang ditandatanganinya.

Dalam ekosistem distribusi perangkat lunak, tanda tangan digital menjadi standar emas untuk memvalidasi keaslian berkas. Berbeda dengan tanda tangan fisik yang bersifat statis (bentuknya selalu sama), tanda tangan digital bersifat dinamis. Bentuk string tanda tangan akan selalu berubah total jika dokumen yang ditandatangani berbeda, meskipun penandatangannya adalah orang yang sama.

Sifat utama yang harus dipenuhi oleh skema tanda tangan digital meliputi:

1. Autentikasi Sumber
   Penerima dapat memverifikasi identitas penanda tangan (pemilik kunci publik) dengan kepastian matematis.
2. Integritas Data
   Tanda tangan harus bergantung pada setiap bit dari pesan. Jika satu bit saja pada berkas *mod* diubah (misalnya disisipkan *malware*), maka verifikasi tanda tangan akan gagal total.
   Tanda tangan pada berkas A tidak bisa dipindahkan ke berkas B.
3. Nirsangkal (*Non-repudiation*)
   Penanda tangan tidak dapat menyangkal di kemudian hari bahwa ia telah menandatangani berkas tersebut, karena tanda tangan tersebut hanya bisa dibuat menggunakan kunci privat yang hanya dimilikinya.
4. Tidak Dapat Dipalsukan (*Unforgeable*)
   Secara komputasi, haruslah tidak mungkin bagi penyerang untuk membuat tanda tangan yang valid untuk suatu pesan tanpa mengetahui kunci privat penanda tangan.

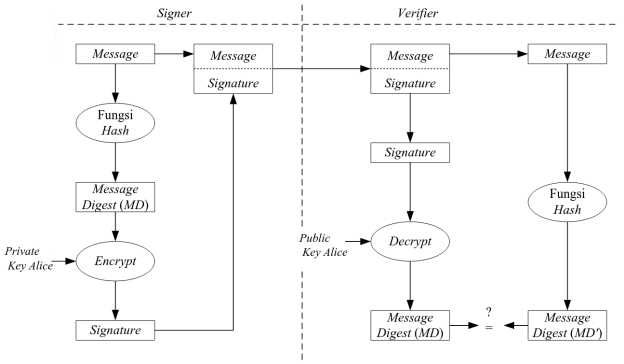
Proses pembangkitan tanda tangan (*signing*) dilakukan oleh pengirim (dalam kasus ini, *modder* atau pembuat *mod*). Langkah-langkah teknisnya adalah sebagai berikut:

1. Peringkasan Data (*Hashing*)
   Berkas *mod* asli (misalnya Mod.zip) diproses menggunakan fungsi *hash* satu arah (seperti SHA-256) untuk menghasilkan nilai ringkasan pesan yang disebut *Message Digest*. Langkah ini penting karena mengenkripsi *digest* jauh lebih efisien daripada mengenkripsi seluruh isi berkas yang berukuran besar.
2. Enkripsi Digest
   Message Digest tersebut kemudian dienkripsi menggunakan Kunci Privat milik pengirim.
3. Pembentukan Tanda Tangan
   Hasil dari enkripsi *digest* inilah yang disebut sebagai Tanda Tangan Digital.
4. Pengemasan Tanda Tangan Digital
   Tanda tangan digital tersebut kemudian disisipkan ke dalam arsip *mod* atau disertakan sebagai berkas terpisah (misalnya .sig) bersamaan dengan berkas *mod* asli untuk didistribusikan ke publik.

Proses verifikasi tanda tangan dilakukan oleh penerima (dalam kasus ini, *gamer* atau pengguna). Tujuannya adalah memastikan bahwa berkas yang diterima aman. Langkah-langkahnya adalah:

1. Pemisahan Berkas
   Sistem pada penerima memisahkan berkas data asli dengan berkas tanda tangan digital yang diterima.
2. Dekripsi Tanda Tangan
   Tanda tangan digital didekripsi menggunakan Kunci Publik pengirim. Proses ini akan mengembalikan nilai *Message Digest* asli (sebut saja Digest A) yang dibuat oleh pengirim. Jika proses dekripsi gagal, berarti tanda tangan tersebut tidak dibuat oleh pasangan kunci privat yang sah.
3. Hashing Ulang
   Sistem pada penerima mengambil berkas *mod* yang diunduh dan menghitung nilai *hash*-nya secara mandiri menggunakan algoritma yang sama (SHA-256). Hasil perhitungan ini menghasilkan Digest B.
4. Komparasi
   Sistem membandingkan Digest A (hasil dekripsi) dengan Digest B (hasil hitung mandiri). Jika Digest A sama dengan Digest B, maka verifikasi berhasil. Hal ini membuktikan dua hal sekaligus: data tidak berubah (integritas) dan pengirimnya valid (autentikasi). Jika Digest A tidak sama dengan Digest B, maka verifikasi gagal. Hal ini menandakan bahwa berkas telah termodifikasi (*corrupt*/disusupi *malware*) atau tanda tangan tersebut palsu.

Gambaran umum mekanisme pembangkitan dan verifikasi tanda tangan digital tercantum pada Gambar 2.1.



Gambar 2.1 Alur Pembangkitan dan Verifikasi Tanda Tangan Digital

D. Algoritma RSA dan Skema PSS

RSA adalah algoritma kriptografi kunci publik yang sangat populer dan digunakan secara luas untuk mengamankan data internet. Algoritma ini dinamai sesuai dengan nama penemunya, yaitu Rivest, Shamir, dan Adleman. Fungsi utama RSA dalam sistem keamanan adalah untuk melakukan enkripsi data dan pembuatan tanda tangan digital.

Keamanan RSA bergantung pada konsep matematika

yang disebut faktorisasi bilangan bulat. Secara sederhana, algoritma ini bekerja dengan cara mengalikan dua bilangan prima yang sangat besar untuk menghasilkan sebuah kunci. Bagi komputer, mengalikan dua angka besar sangatlah mudah. Namun, kebalikannya, yaitu memecah hasil perkalian tersebut kembali menjadi dua angka prima asalnya, adalah tugas yang sangat sulit dan membutuhkan waktu ribuan tahun jika dilakukan oleh komputer biasa. Inilah yang membuat RSA aman, yaitu orang lain bisa mengetahui hasil perkaliannya (kunci publik), tetapi tidak bisa menebak angka asalnya (kunci privat).

Untuk standar keamanan modern saat ini, lembaga standar teknologi NIST (National Institute of Standards and Technology) menyarankan penggunaan ukuran kunci minimal 2048-bit. Ukuran ini dianggap cukup kuat untuk menahan serangan komputer canggih sekalipun hingga beberapa tahun ke depan.

Dalam penerapannya, algoritma RSA tidak bekerja sendirian. RSA membutuhkan sebuah metode tambahan yang disebut *padding* untuk memastikan data aman dari pola-pola tertentu yang bisa dibaca peretas. Salah satu metode *padding* yang aman saat ini adalah PSS atau *Probabilistic Signature Scheme*.

Berbeda dengan metode lama yang bersifat deterministik (pasti/tetap), PSS bersifat probabilistik (berpeluang/acak). Pada metode lama, jika seseorang menandatangani dokumen yang sama dua kali, hasil tanda tangan digitalnya akan selalu terlihat sama persis. Hal ini kurang aman karena polanya bisa dipelajari.

Sebaliknya, PSS menambahkan elemen acak yang disebut *salt* ke dalam proses pembuatan tanda tangan. Akibatnya, jika kita menandatangani dokumen yang sama sebanyak sepuluh kali, PSS akan menghasilkan sepuluh tanda tangan digital yang bentuknya berbeda-beda. Meskipun bentuknya berbeda, semua tanda tangan tersebut tetap valid dan bisa diverifikasi menggunakan kunci publik yang sama. Sifat acak inilah yang membuat tanda tangan digital dengan skema RSA-PSS jauh lebih sulit dipalsukan dibandingkan metode terdahulu.

E. Fungsi Hash SHA-256

Algoritma SHA-256 (*Secure Hash Algorithm* 256-bit) adalah fungsi *hash* kriptografis yang dikembangkan oleh NSA. Fungsi ini memetakan data *input* dengan panjang berapapun menjadi *output* (*digest*) dengan panjang tetap 256 bit (32 byte). Karakteristik SHA-256 yang membuatnya cocok untuk verifikasi *mod* adalah:

1. Resistensi Kolisi (*Collision Resistance*)
   Sangat sulit (hampir mustahil secara komputasi) untuk menemukan dua berkas *mod* berbeda yang menghasilkan nilai *hash* yang sama.
2. Efek *Avalanche*
   Perubahan kecil pada input (misalnya mengubah satu baris kode pada *mod*) akan mengubah nilai *hash* output secara drastis dan menyeluruh. Hal ini memastikan bahwa modifikasi *malware* sekecil apapun akan terdeteksi.

F. Cities: Skylines II

Cities: Skylines II adalah permainan simulasi pembangunan kota yang dikembangkan oleh Colossal Order dan diterbitkan oleh Paradox Interactive. Dirilis pada tahun 2023, gim ini merupakan sekuel dari Cities: Skylines yang telah menjadi standar industri dalam genre simulasi. Permainan ini menawarkan skala pembangunan yang jauh lebih masif dibandingkan pendahulunya, memungkinkan pemain untuk merancang kawasan perkotaan dari kota kecil hingga menjadi metropolis raksasa dengan tingkat realisme yang tinggi.

Inti dari permainan ini menempatkan pemain sebagai pengelola kota yang bertanggung jawab atas perencanaan tata ruang (*zoning*), pembangunan infrastruktur vital, hingga manajemen layanan publik. Salah satu keunggulan teknis utama dari seri ini adalah implementasi simulasi kecerdasan buatan yang mendalam pada setiap penduduknya. Setiap warga memiliki siklus hidup, pekerjaan, dan pola perjalanan yang unik, menciptakan dinamika lalu lintas dan simulasi ekonomi yang sangat kompleks serta responsif terhadap keputusan pemain.

Secara teknis, gim ini dibangun di atas mesin permainan (*game engine*) Unity yang memberikan fondasi kuat untuk menangani ribuan agen simulasi secara bersamaan. Arsitektur ini memungkinkan perhitungan *real-time* terhadap berbagai variabel kota, mulai dari arus lalu lintas yang dinamis, perubahan cuaca, hingga fluktuasi ekonomi mikro. Kompleksitas sistem ini menuntut kinerja komputasi yang tinggi untuk menjaga kestabilan simulasi saat kota berkembang menjadi semakin padat dan kompleks.

G. Mod Cities: Skylines II

Modifikasi atau *mod* pada Cities: Skylines II merupakan fitur eksternal yang memungkinkan pemain untuk memperluas fungsionalitas dan mekanisme permainan melampaui batasan aslinya. *Mod* bekerja dengan cara memanipulasi kode inti permainan untuk mengubah perilaku simulasi, seperti mengatur ulang sistem algoritma lalu lintas, mengubah perhitungan ekonomi kota, atau menambahkan alat bantu pembangunan (*building tools*) yang baru. Salah satu fungsionalitas tambahan yang disediakan oleh *mod* diilustrasikan pada Gambar 2.2. Secara struktur, mod ini umumnya terdiri dari berkas kode program (seperti .dll) beserta berkas aset pendukung lainnya yang dikemas menjadi satu kesatuan dalam arsip berformat ZIP.

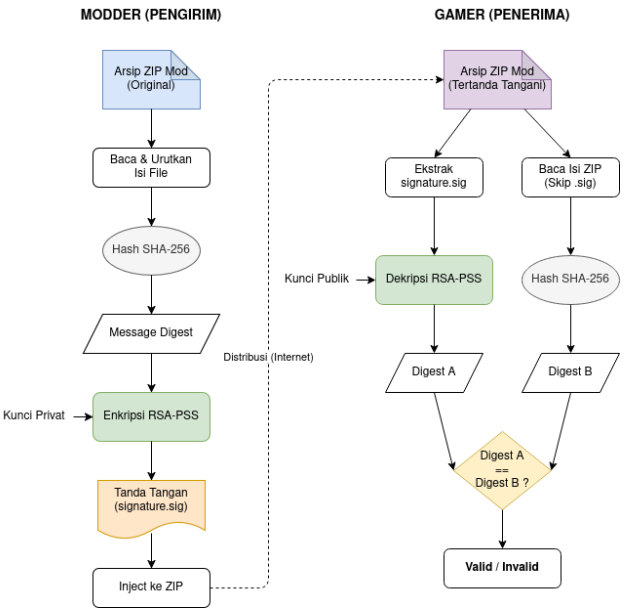
Gambar 2.2 Fungsionalitas Tambahan: *Lane Connection Tool* yang Disediakan oleh *Mod* bernama "Traffic"

Kemudahan dalam pembuatan dan distribusi *mod* ini telah mendorong terbentuknya komunitas kreatif yang aktif membagikan karya mereka melalui internet, baik melalui platform resmi Paradox Mods maupun situs komunitas pihak ketiga. Pengguna cukup mengunduh arsip *mod* tersebut dan menempatkannya ke dalam direktori permainan untuk mengaktifkan fitur-fitur baru tersebut. Namun, karena modifikasi ini melibatkan eksekusi kode program secara langsung di dalam memori komputer saat permainan berjalan, berkas *mod* memiliki akses yang luas terhadap sistem. Hal ini menjadikan aspek keamanan dari sumber berkas *mod* sebagai poin krusial dalam Cities: Skylines II.

III. Rancangan Skema Tanda Tangan Digital

Modifikasi (*mod*) untuk gim Cities: Skylines II umumnya didistribusikan dalam bentuk arsip terkompresi berformat ZIP yang memuat berkas kode biner (.dll) serta aset pendukung lainnya. Dalam perancangan sistem ini, proses hashing tidak dilakukan terhadap *file* ZIP sebagai satu kesatuan *file* biner, melainkan terhadap konten hasil pembacaan arsip di dalamnya. Pendekatan ini dipilih untuk menjamin konsistensi nilai *hash* antara pengirim dan penerima, terlepas dari metadata kompresi yang mungkin berbeda di setiap sistem operasi.

Algoritma *hash* yang digunakan adalah SHA-256, yang dipilih karena memiliki resistensi kolisi yang kuat. Untuk proses enkripsi dan dekripsi tanda tangan, sistem menggunakan algoritma asimetris RSA dengan panjang kunci 2048-bit. Berbeda dengan skema RSA klasik, sistem ini menerapkan skema padding PSS (*Probabilistic Signature Scheme*) untuk meningkatkan keamanan kriptografis terhadap serangan pola tertentu. Agar tidak terjadi *circular dependency* saat verifikasi, berkas tanda tangan (signature.sig) yang dihasilkan akan disimpan di dalam arsip ZIP tersebut, namun secara eksplisit akan diabaikan atau dilewati ketika sistem melakukan perhitungan *hash* di sisi penerima.



Gambar 3.1 Skema Penandatanganan dan Verifikasi *Mod*

Langkah-langkah detail proses penandatanganan (*signing*) yang dilakukan oleh pihak pengirim (*modder*) adalah sebagai berikut:

1. Pengirim menyiapkan seluruh berkas *mod* (kode dan aset) dan mengemasnya ke dalam sebuah arsip ZIP.
2. Program akan membaca seluruh struktur berkas di dalam ZIP tersebut. Penting untuk dicatat bahwa program akan mengurutkan daftar berkas di dalam arsip ZIP secara alfabetis (A-Z) terlebih dahulu untuk memastikan urutan pembacaan data yang deterministik (pasti) dan konsisten.
3. Program menghitung nilai *hash* SHA-256 dari seluruh konten berkas yang telah diurutkan tersebut untuk menghasilkan sebuah *message digest* yang unik.
4. *Message digest* tersebut kemudian dienkripsi menggunakan Kunci Privat milik pengirim dengan algoritma RSA-PSS. Hasil enkripsi ini adalah tanda tangan digital.
5. Tanda tangan digital tersebut kemudian disisipkan kembali ke dalam arsip ZIP yang sama dengan nama signature.sig, sehingga file *mod* siap didistribusikan.

Langkah-langkah verifikasi yang dilakukan oleh pihak penerima (pengguna/*gamer*) adalah sebagai berikut:

1. Penerima mengunduh atau menerima arsip ZIP *mod* yang di dalamnya memuat tanda tangan digital.
2. Program akan mencari dan mengekstrak berkas signature.sig dari dalam arsip, kemudian mendekripsinya menggunakan Kunci Publik pengirim. Proses ini akan menghasilkan *message digest* asli (Digest A).
3. Secara bersamaan, program akan membaca kembali seluruh isi konten arsip ZIP dan mengurutkannya secara alfabetis. Program akan menghitung *hash* SHA-256 dari konten tersebut dengan aturan khusus, yaitu mengabaikan/melewati berkas signature.sig. Proses ini menghasilkan *message digest* baru (Digest

B).

4. Sistem membandingkan Digest A (hasil dekripsi) dan Digest B (hasil hitung mandiri). Apabila kedua nilai tersebut identik, maka verifikasi dinyatakan berhasil, yang membuktikan bahwa *mod* tersebut asli dan tidak disusupi *malware*. Sebaliknya, jika berbeda, *mod* dianggap berbahaya.

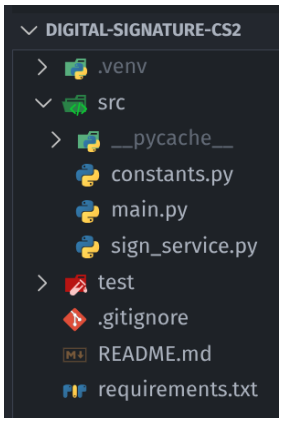
IV. Analisis dan Pembahasan

A. Implementasi Sistem

Pada implementasi ini, penulis menggunakan bahasa pemrograman Python versi 3.10.12. Pemilihan bahasa ini didasarkan pada ketersediaan pustaka kriptografi yang matang serta kemudahan dalam pengelolaan berkas arsip. Penulis memanfaatkan pustaka eksternal bernama cryptography versi 46.0.3 untuk menangani operasi algoritma RSA, PSS, dan SHA-256. Selain itu, pustaka bawaan zipfile digunakan untuk memanipulasi arsip *mod* dan argparse digunakan untuk membangun antarmuka baris perintah (*Command Line Interface*).

Program klien yang dibangun memiliki tiga fitur utama yang dapat diakses melalui terminal, yaitu:

1. Pembangkitan Kunci (*Key Generation*)
   Fitur untuk membangkitkan sepasang kunci (privat dan publik) bagi pengembang *mod*. Kunci disimpan dalam format PEM.
2. Penandatanganan Mod (*Signing*)
   Fitur untuk memproses arsip ZIP *mod*, menghitung *hash* dari seluruh isinya, dan menyisipkan tanda tangan digital ke dalam arsip tersebut.
3. Verifikasi Mod (*Verification*)
   Fitur untuk memeriksa keaslian arsip *mod* menggunakan kunci publik pengembang.



Gambar 4.1 Struktur Kode Program

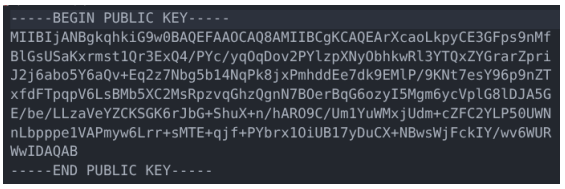
Dalam struktur program (ditunjukkan pada Gambar 4.1), logika kriptografi dipisahkan dalam modul sign_service.py, sedangkan interaksi pengguna ditangani oleh main.py. Hal ini memudahkan pengembangan dan pemeliharaan kode.

B. Pengujian dan Analisis

Pengujian dilakukan dengan menggunakan sebuah *mod* Cities: Skylines II bernama "Traffic". *Mod* ini diasumsikan berisi *file* kode biner (.dll) serta *file* konfigurasi lainnya yang tersimpan dalam sebuah *file* arsip ZIP. Penulis melakukan beberapa skenario pengujian untuk memvalidasi keamanan sistem.

1. Pembangkitan Kunci
   Sebelum melakukan penandatanganan, penulis membangkitkan pasangan kunci privat-publik terlebih dahulu. Menggunakan perintah generate-keys, program berhasil membuat dua berkas kunci: private_key.pem (rahasia) dan public_key.pem (untuk disebarkan). Proses pembangkitan kunci dan hasil kuncinya dapat dilihat pada Gambar 4.2, 4.3, dan 4.4.

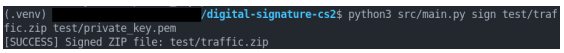
Gambar 4.2 Tangkapan Layar Proses Pembangkitan Kunci



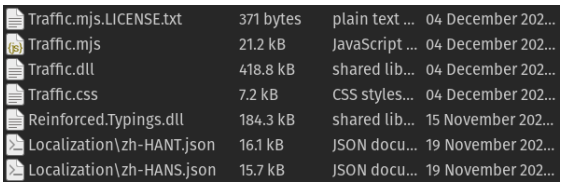
Gambar 4.3 Tangkapan Layar Kunci Publik

Gambar 4.4 Tangkapan Layar Kunci Privat

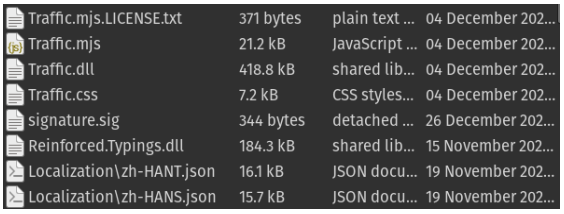
2. Penandatanganan Arsip Mod
   Penulis kemudian melakukan penandatanganan terhadap berkas "traffic.zip" menggunakan kunci privat yang telah dibuat. Program membaca seluruh isi arsip, mengurutkan nama berkas untuk konsistensi, dan menghasilkan berkas tanda tangan bernama signature.sig yang dimasukkan kembali ke dalam arsip ZIP.



Gambar 4.5 Tangkapan Layar Proses Penandatanganan *Mod*



Gambar 4.6 Tangkapan Layar Isi ZIP Sebelum Ditandatangani



Gambar 4.7 Tangkapan Layar Isi ZIP Setelah Ditandatangani (terdapat *file* signature.sig)

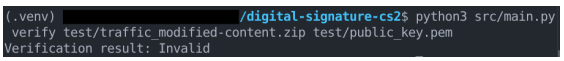
Dapat dilihat pada Gambar 4.5, 4.6, dan 4.7, arsip *mod* kini memiliki tambahan berkas signature.sig yang berisi data kriptografis untuk verifikasi.

3. Verifikasi Valid (*Mod* Asli)
   Skenario pertama verifikasi dilakukan terhadap mod yang belum dimodifikasi sama sekali. Penulis menjalankan perintah *verify* menggunakan kunci publik yang sesuai.

Gambar 4.8 Tangkapan Layar Hasil Verifikasi Valid

Hasil verifikasi dapat dilihat pada Gambar 4.8. Hasil verifikasi tersebut menunjukkan bahwa integritas data terjaga dan tanda tangan digital valid sesuai dengan pasangan kunci yang digunakan.

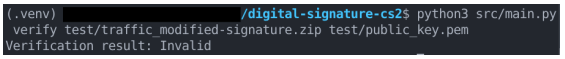
4. Verifikasi Gagal (Modifikasi Konten)
   Skenario ini menguji ketahanan sistem terhadap serangan integritas *file*. Penulis melakukan manipulasi dengan cara mengekstrak arsip *mod*, menghapus salah satu berkas di dalamnya, lalu mengemasnya kembali menjadi ZIP tanpa memperbarui tanda tangan. Saat dilakukan verifikasi ulang, sistem mendeteksi ketidakcocokan tanda tangan.



Gambar 4.9 Tangkapan Layar Hasil Verifikasi Gagal (Modifikasi Isi *Mod*)

Hasil verifikasi dapat dilihat pada Gambar 4.9. Hal ini membuktikan bahwa perubahan sekecil apapun pada isi arsip, termasuk penghapusan atau penyisipan berkas berbahaya, akan mengubah nilai *message digest* secara keseluruhan sehingga verifikasi RSA-PSS akan gagal.

5. Verifikasi Gagal (Tanda Tangan Palsu/Rusak)
   Skenario terakhir dilakukan dengan memodifikasi isi berkas signature.sig secara manual, misalnya mengubah satu karakter pada string Base64. Saat diverifikasi, program juga mengembalikan hasil Invalid. Hasil verifikasi dapat dilihat pada Gambar 4.10.



Gambar 4.10 Tangkapan Layar Hasil Verifikasi Gagal (Tanda Tangan Salah)

V. KESIMPULAN

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, dapat disimpulkan bahwa skema tanda tangan digital menggunakan algoritma RSA dengan *padding* PSS (*Probabilistic Signature Scheme*) dan fungsi *hash* SHA-256 berhasil diterapkan untuk mengamankan distribusi *mod* Cities: Skylines II. Program yang dibangun mampu memverifikasi integritas arsip *mod* secara akurat, di mana sistem berhasil membedakan antara *mod* asli yang ditandatangani oleh pengembang resmi dan *mod* yang telah dimanipulasi isinya (seperti penyisipan kode berbahaya pada berkas .dll). Dengan mekanisme ini, pengguna *mod* memiliki cara mandiri untuk memvalidasi bahwa arsip *mod* yang mereka unduh benar-benar berasal dari pembuat yang sah dan tidak mengalami perubahan di tengah jalan, sehingga risiko eksekusi kode sembarangan (*arbitrary code execution*) dapat dimitigasi.

Untuk pengembangan lebih lanjut, penulis menyarankan beberapa hal berikut:

1. Penerapan *Public Key Infrastructure*
   Implementasi saat ini masih menggunakan distribusi kunci publik secara manual. Disarankan adanya *Public Key Infrastructure* (PKI) atau Otoritas Sertifikat (CA) terpusat agar distribusi dan validasi kunci publik dapat dilakukan secara otomatis dan terpercaya.
2. Integrasi *Launcher*
   Penulis menyarankan agar mekanisme verifikasi ini tidak hanya berjalan sebagai alat CLI (*Command Line Interface*) terpisah, melainkan diintegrasikan langsung ke dalam *Mod Manager* atau *Launcher* gim. Hal ini akan memudahkan pengguna awam untuk mendapatkan perlindungan keamanan secara transparan tanpa perlu menjalankan perintah manual.
3. Antarmuka Grafis (GUI)
   Pembuatan antarmuka pengguna berbasis grafis akan sangat membantu meningkatkan aksesibilitas alat ini bagi komunitas *modding* yang lebih luas.

VI. Tautan Program

Kode program lengkap yang mencakup implementasi pembangkitan kunci, penandatanganan, dan verifikasi mod yang digunakan dalam makalah ini dapat diakses melalui tautan repositori berikut:
https://github.com/bastianhs/digital-signature-cs2

VII. Ucapan Terima Kasih

Penulis ingin menyampaikan rasa terima kasih yang sebesar-besarnya kepada semua pihak yang telah membantu dan memberikan dukungan, baik secara langsung maupun tidak langsung, dalam penyusunan makalah yang berjudul "Implementasi Tanda Tangan Digital untuk Verifikasi Mod Cities: Skylines II". Makalah ini disusun sebagai pemenuhan tugas akhir mata kuliah IF4020 Kriptografi.

Secara khusus, penulis ingin mengucapkan terima kasih kepada:

1. Tuhan Yang Maha Esa, karena atas berkat, rahmat, dan karunia-Nya, penulis dapat menyelesaikan makalah ini dengan baik dan tepat pada waktunya.
2. Bapak Dr. Ir. Rinaldi, M.T., selaku dosen pengajar mata kuliah IF4020 Kriptografi yang telah memberikan ilmu, wawasan, serta materi pembelajaran yang sangat komprehensif dan bermanfaat bagi pengerjaan makalah ini.
3. Kedua orang tua dan keluarga penulis yang senantiasa memberikan doa, semangat, serta dukungan moral kepada penulis.
4. Teman-teman penulis yang telah banyak memberikan masukan, diskusi yang membantu, dan motivasi selama proses pengerjaan makalah ini.

Penulis menyadari bahwa makalah ini masih jauh dari kata sempurna. Oleh karena itu, penulis sangat terbuka terhadap kritik dan saran yang membangun demi perbaikan di masa mendatang.

Referensi

[1] Ahmad, J. I., Din, R., & Ahmad, M. (2018). Analysis Review on Public Key Cryptography Algorithms. *Indonesian Journal of Electrical Engineering and Computer Science*, *12*(2), 447–454. https://doi.org/10.11591/ijeecs.v12.i2.pp447-454

[2] Gawali, S. B. (2023). A Comprehensive Study on Digital Signatures. International Journal of Advanced Research in Science, Communication and Technology, 37–39. https://doi.org/10.48175/ijarsct-11608

[3] Kaliski, B. (2003). Raising the Standard for RSA Signatures: RSA-PSS. https://www.gradenegger.eu/download/pdf/rsa-pss.pdf. Diakses pada 26 Desember 2025.

[4] Knight, K. (2024). Cities: Skylines 2 Players May Have Been Infected by Malware. https://gamerant.com/cities-skylines-2-traffic-mod-virus. Diakses pada 26 Desember 2025.

[5] Paradox Interactive. (2024). Important Update Regarding Traffic Mod. https://www.paradoxinteractive.com/games/cities-skylines-ii/news/traffic-breach-statement. Diakses pada 26 Desember 2025.

[6] Rinaldi. (2025). Algoritma RSA. https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/20-Algoritma-RSA-2025.pdf. Diakses pada 26 Desember 2025.

[7] Rinaldi. (2025). Fungsi Hash. https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/26-Fungsi-hash-2025.pdf. Diakses pada 26 Desember 2025.

[8] Rinaldi. (2025). Kriptografi Kunci-Publik. https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/19-Kriptografi-Kunci-Publik-2025.pdf. Diakses pada 26 Desember 2025.

[9] Rinaldi. (2025). Pengantar Kriptografi. https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/01-Pengantar-Kriptografi-(2025).pdf. Diakses pada 26 Desember 2025.

[10] Rinaldi. (2025). Tanda-tangan Digital. https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/29-Tanda-tangan-digital-2025.pdf. Diakses pada 26 Desember 2025.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 Desember 2025

Bastian H. Suryapratama
13522034